INFORMATION SYSTEMS SERVICES

Getting Started With The UNIX Operating System

This document provides an introduction to the UNIX operating system.

AUTHOR: Information Systems Services

DATE: August 2005

EDITION: 1.8

BEG 8

UNIVERSITY OF LEEDS

Contents

Further Information

Further information about the UNIX operating system is given in the following documents:

Introduction To UNIX: Exercises (TUT 5)
Further Exercises In UNIX: (TUT 14)
UNIX Facts Sheet (FAC 2)
vi Facts Sheet (FAC 3)
GNU Emacs Facts Sheet (FAC 4)
Pascal Facts Sheet (FAC 5)
C Compiler Facts Sheet (FAC 6)
Elm Facts Sheet (FAC 7)
Fortran Facts Sheet (FAC 8)

Format Conventions

In this document the following format conventions are used:

Commands that you must type in are shown in **`bold Courier`** font.	**`WIN31`**
Menu items are given in a **Bold, Arial** font.	**Windows Applications**
Keys that you press are enclosed in angle brackets.	**<Enter>**

Feedback

If you notice any mistakes in this document please contact the Information Officer. Email should be sent to the address **`info-officer@leeds.ac.uk`**

Print Record

This document was printed on 17-Aug-05.

1 Introduction

Purpose Of This Document

This document is intended for people who are new to the UNIX operating system. The document aims to provide an understanding of basic UNIX concepts and to provide sufficient knowledge for you to try out some simple UNIX commands.

Most of the material in this document will be valid if you use any UNIX computer system, provided you use the C shell. The C shell environment will be provided on all UNIX computer systems administered by the ISS. If you are using a departmental UNIX system, you should consult with your local computer support staff to check that the material given in this document is applicable on your system.

Local Enhancements To UNIX

The ISS will enhance UNIX by providing locally developed commands which will make UNIX easier to use. However locally developed commands may not be available on departmental machines and will not be described in manufacturer's manuals or other publications. In this document locally developed commands will be indicated.

Practicing UNIX Commands

This document contains a number of exercises which will help you gain experience in the use of UNIX. Exercises are including on the following pages:

Exercise 1	Logging in	8
Exercise 2	Simple filestore commands	15
Exercise 3	Changing file permissions	20
Exercise 4	The help system	21
Exercise 5	Input and output redirection	23
Exercise 6	Piping	24

2 The UNIX Operating System

What is UNIX?

UNIX is an operating system which is available on a wide range of computer systems, including personal computers, workstations, mainframe systems and supercomputers.

Why UNIX?

UNIX has many advantages over other (proprietary) operating systems:

• Portable	UNIX is written in the high level language C. This makes it easy to install on new computing systems.
• Mainstream	UNIX is available on many computer systems including Sun and Silicon Graphics. It is particularly popular for systems used within research environments.
• Popular	A wide range of applications software is available on the UNIX operating system.
• Powerful	UNIX has many powerful features, particularly for program development.
• Standardisation	Although in the past many different versions of UNIX have been developed, various groups are currently attempting to define a standard UNIX.

Main Features of UNIX

The Kernel and the Shell

The UNIX operating system consists of a *kernel* (which carries out basic operating system functions such as accessing files, etc.) and a *shell* which provides the command line user interface to the kernel. A number of shells are available on the UNIX operating system including:

- Bourne shell
- C shell
- Korn shell
- Bash shell

The *Bourne shell* is one of the oldest shells and is the most efficient for background work. However it provides few facilities for interactive users.

The *C shell* provides sophisticated interactive capabilities lacking in the Bourne shell. Features of this shell include a command history buffer, command aliases and file name completion. The C shell has a syntax which resembles the C programming language.
The C shell is the default shell for interactive work on many UNIX systems. It will be covered in this document, although most of the

basic commands given here are relatively standard across all the main shells.

The *Korn shell* was written by David Korn from AT&T and in it he attempted to merge the preferred features of both the Bourne and C shells as well as adding some additional features.
Unfortunately the *Korn* shell was not available for free, as other UNIX shells were, so many users and companies did not chose to use of it.

The *Bash shell* was based on the *Bourne shell* (Bourne again shell) and as with Korn it attempted to combine the best features of the other shells which were available at the time. This shell however was available for free.
Bash was initially adopted for LINUX although several varieties of LINUX now exist e.g. RedHat, SuSE and Debian-GNU

Graphical User Interfaces

Graphical User Interfaces (also known as GUIs) provide an alternative user interface to the C and Bourne shell. GUIs provide a replacement to the command line interface based on the use of menus and a mouse. Using GUIs, applications software from different suppliers can have a consistent interface, which reduces the time needed to master new applications.

3 Accessing UNIX Systems

A number of ways are available for accessing UNIX systems. Four recommended options are described below.

1. Direct access to a UNIX Workstation

In order to use a UNIX workstation, such as a Sun or Silicon Graphics workstation, you simply need to sit in front of the workstation and use the attached keyboard. If the screen is blank you may need to move the mouse or press a key in order to activate the screen display. You will then need to give your username and password as described in section 4.

2. Using a PC as a Simple Command Line Terminal

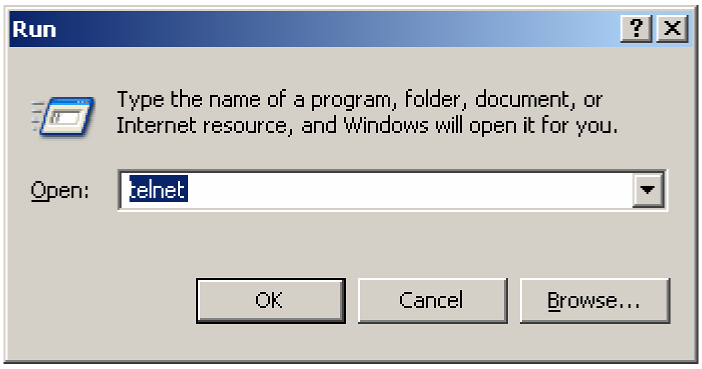
You can access UNIX systems using terminal emulation software on a PC. Terminal emulation software runs a set of protocols (rules) which govern network access to other computer systems.
Telnet is a straight forward terminal emulation program available on our cluster PCs which you can use to access UNIX systems.
To run Telnet you should log in to a PC and follow the procedure given below.

Click on the **Start** button in the lower-left-hand side of the screen

From the **Start** menu select **Run ...**

In the **Run** dialog box delete the command next to **Open :** and type in **telnet**

e.g.



Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

Open: telnet

Click on the **OK** button
The telnet window will open and next to the prompt **Telnet >**

type **open** *hostname*
and press **<Enter>**

e.g. `Microsoft telnet > open imap2`

You will then need to give your username and password as described in section 4.

3. Using TeraTerm Pro - A More Sophisticated Terminal Emulator Program

TeraTerm Pro is similar to telnet but it supports "extensions" to the basic telnet functionality.
To run TeraTerm Pro you should log in to a PC and follow the procedure given below.

Click on the **Start** button in the lower-left-hand side of the screen

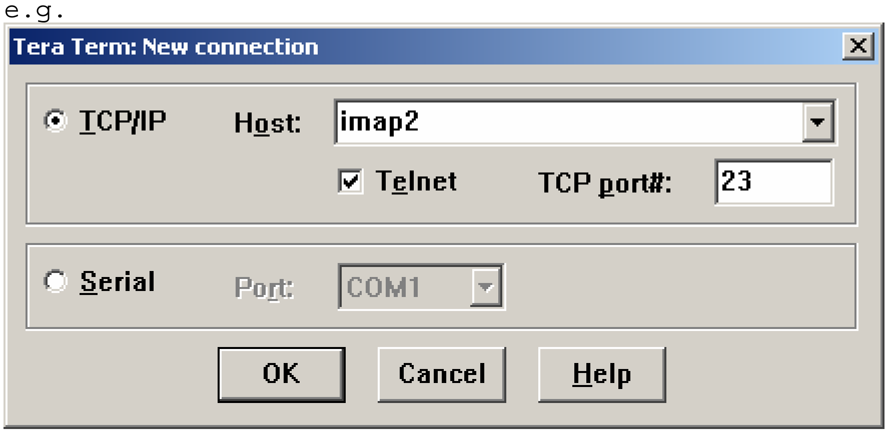
From the **Start** menu select **Programs**

From the **Programs** menu select **Network Tools**

From the **Network Tools** menu select **TeraTerm Pro**

Next to the **Host:** prompt in the dialog box, type the name of the host you wish to connect to

`e.g.`



Click on the **OK** in the dialog box.

TeraTerm will open a connection to the requested host and you will see the login prompt

`login:`

You will then need to give your username and password as described in section 4.

4. Using A PC As An X Windows Terminal

If you have a PC with **Vista eXceed** X server software installed, your PC can be enabled to use a Graphical User Interface (GUI) such as Motif. Applications software which have been written for GUIs are relatively easy to use.

Further information in the use of Vista eXceed software is given in the document *Getting Started With Vista eXceed* (BEG 15).

PCs available in ISS clusters have Vista eXceed available from the desktop. You can follow the actions given below to open a **Vista eXceed** session

Click on the **Start** button in the lower-left-hand side of the screen

From the **Start** menu select **Programs**

From the **Programs** menu select **Network Tools**

From the **Network Tools** menu select **Xstart**

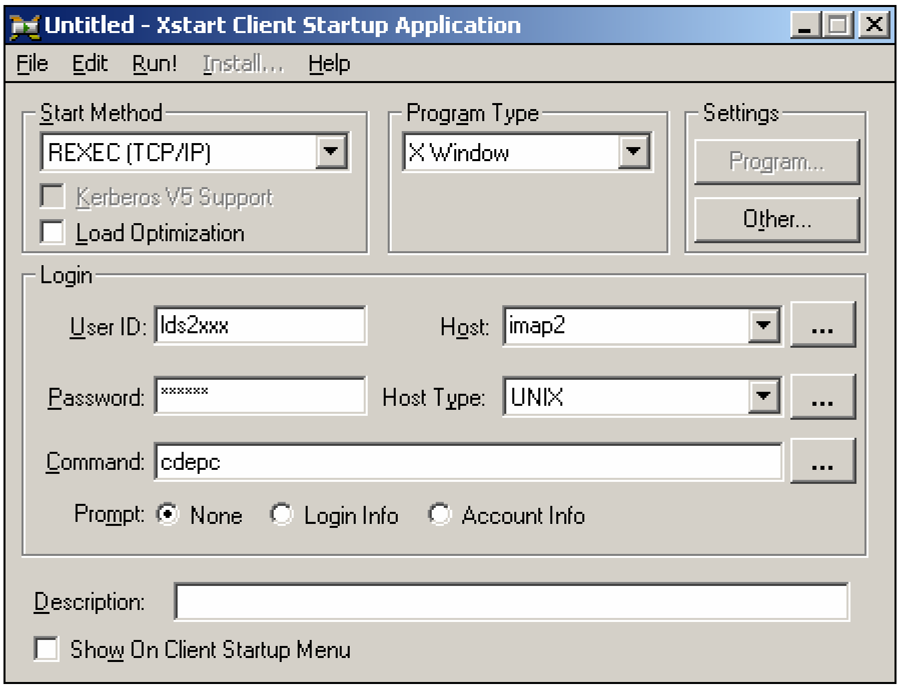
In the dialog box, type the required information next to the following prompts.

UserID: `username`	**Host:** `hostname`
Password: `password`	**Host Type:** `host_type`

Command: cdepc

Please note that for security reasons, the password is not displayed when you type it in.

See following example



Start Method: REXEC (TCP/IP)
Program Type: X Window
User ID: lds2xxx
Host: imap2
Password: ******
Host Type: UNIX
Command: cdepc
Prompt: [✓ None] Login Info Account Info
Description:

Once all the required information has been entered click on the **Run!** option on the top menu bar of the dialog box.

After a little time **Vista eXceed** will open a connection to the requested host and log you in to the host so you can now skip the first part of Section 4.

Note **cdepc** is the program that you request to be run on the host you are connecting to (in this example imap2). It is this program that enables the unix host to communicate back to *Vista eXceed* on your PC and hence enables the GUI environment to be viewed on the PC.
If you have difficulty getting *Vista eXceed* to connect to the requested host it may be because the host does not have **cdepc** available. In this case you should contact the system administrator of the required host to find out if an alternative command can be used instead.

4 Logging In, Logging Out and Changing Your Password

Logging In

When you have established contact with the UNIX system, the login prompt will be displayed. You must then give your username followed by your password:

```
Login: men5jb
Password: ru4jam
```

NOTE *For security reasons the password is not displayed*

The *username* can be up to 8 characters in length. UNIX usernames contain only lowercase characters.

The *password* must normally contain between 6 and 8 characters. On some UNIX systems the password must contain at least 1 non alphabetic character.

men5jb and *ecl6bk* are valid usernames.

rsmkirkwood is not a valid username (too long).

ru4jam is a valid password.

me2 is not a valid password (too short).

System Messages

When you login a number of system messages may be displayed. The message:

```
You have mail
```

indicates that electronic mail has been sent to your mailbox. The message:

```
News:101 courses
```

indicates that a news file called *courses* (number 101) which contains information on ISS courses is available for viewing. Use the **news** command to look at it.

```
News:101 courses
% news 101
Courses
A UNIX course will be held on 15 October.
```

At times the news system will be used to provide important pieces of information. You are advised to use the **news** command on a regular basis.

Changing Your Password

Your username and initial password are assigned by the System Administrator. You are advised to change the initial password using the **passwd** command as shown below:

```
% passwd
Enter login(NIS) password: oldpassword
New password:  newpassword
Retype new password:  newpassword
%
```

Logging Out

When you have finished your UNIX session you should logout from the system. To do this give the command:

```
% logout
```

You should always wait for a message confirming that you have logged out.

If you have been using a PC to gain access to the UNIX system remember to log off from the PC as well.

Problems? You may occasionally get the message:

```
There are stopped jobs
```

If this happens simply give the **logout** command again.

On some UNIX systems you may receive the message:

```
(logout): command not known
```

If this happens you should type:

```
% exit
```

Exercises

Within this document you will find a series of exercises in grey boxes as below. These exercises are designed to allow you to practice your understanding of UNIX.
The exercises may be attempted, either as part of a course, or on your own. If you have any problems, please consult your supervisor or the Help Desk.

Exercise 1

1. Login to the UNIX system using your username and password.
2. Change your password using the **passwd** command.
3. Logout.
4. Login again using the new password.

5 The UNIX Filestore

UNIX has a hierarchical tree-like filestore. The filestore contains files and directories, as illustrated in the diagram below.

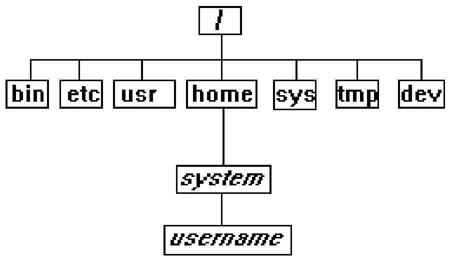


Figure 1 An Illustration of the UNIX Filestore Hierarchy.

The top-level directory is known as the *root*. Beneath the root are several system directories. Directories are identified by the / character. For example:

```
/bin          /etc          /usr
```

Historically, user directories were often kept in the directory `/usr`. However, it is often desirable to organise user directories in a different manner. For example:

`/home/sunserv1`	users on the Sun
`/home/ecuserv1`	users on Engineering department's Suns

Users have their own directory in which they can create and delete files, and create their own sub-directories. For example:

`/home/ecuserv1/men5jb`	belongs to Jane Brown, a postgraduate student in Mechanical Engineering

Directories will include:

`/bin`	Contains many of the programs which will be executed by users
`/etc`	Files used by system administrators
`/dev`	Hardware peripheral devices
`/lib`	System libraries
`/tmp`	Directory for temporary files
`/usr`	Normally contains applications software
`/home/`*`system`*	User directories on UNIX *system* where *system* might be *sun*, *sgi*, *gps* etc.

Pathnames

Files and directories may be referred to by their *absolute pathname*. For example:

```
/home/ecuserv1/men5jb/progs/prog1.f
```

The initial **/** signifies you are starting at the root and hence indicates an absolute pathname.

Files and directories may also be referred to by a *relative pathname*. For example:

```
progs/prog1.f
```

relative to the current location.

The Home Directory

Each user has a *home directory*. They will be attached to this directory when they login. Jane Brown's home directory is:

```
/home/ecuserv1/men5jb
```

The symbol ~ can be used to refer to the home directory. If Jane Brown wishes to refer to her file she can give:

```
~/progs/prog1.f
```

rather than typing the long form:

```
/home/ecuserv1/men5jb/progs/prog1.f
```

The symbol ~ can also refer to the home directory of other users.
~cen6js refers to John Smith's home directory.
So Jane can refer to a file in John Smith's home directory using:

```
~cen6js/test.dat
```

The Current Directory

The *current directory* can be referred to by the **.** character (a full stop). This refers to your actual location in the filestore hierarchy. When you log in the current directory is set to your home directory.

The Parent Directory

The *parent directory* is the directory immediately above the current directory. The parent directory can be referred to by the **..** characters (two full stops). For example to refer to the file `test.dat` in the parent directory:

```
../test.dat
```

Relative path names may also be constructed by progressively stepping back through parent directories using the **..** construct.
For example if user men5jb is currently attached to the directory
/home/ecuserv1/men5jb/progs
and user cen6js has a file called example.data in their home directory **/home/ecuserv1/cen6js**
Then user men5jb may refer to that file with the relative pathname
../../cen6js/example.dat

6 UNIX Commands

UNIX commands have the general format:

```
command [options] [item]
```

Note that:

- Commands are case sensitive. The command **`ls`** is different from **`LS`**. In fact **`LS`** is not recognised as a valid command.

- Command options normally consist of a single character. e.g. The **a** option signifies **all** but use the command:

  `ls -a` not **`ls -all`**

- Command options can be combined or listed separately. For example:

  `ls -al` or **`ls -a –l`**

  The **`l`** (letter ell) option signifies **long listing**

- The command `item` is given last. This is very often a file name. For example:

  `ls -l file1.f` not **`ls file1.f -l`**

Examples Of Simple UNIX Commands

The command **`who`** gives a list of logged on users:

```
% who
root     console Nov 11 08:38
ecl6ql   ttyp0   Nov 11 13:01   (129.11.5.140)
root     ttyp1   Nov 11 08:11   (sun075)
csc2u0ev ttyp3   Nov 11 13:05   (csvax1)
cbl6nd   ttyp5   Nov 11 09:57   (cblslcd)
eclsc    ttyp6   Nov  9 16:32   (sgi006)
csc6agc  ttyp8   Nov 11 10:59   (csparc6)
phy6paj  ttypb   Nov 11 11:39
ecl6bk   ttyU2   Nov 11 13:11
```

The command **`finger`** gives the full name of logged in users:

```
% finger
Login       Name              TTY Idle    When
Where
root     System Administrator  co 4:34 Mon 08:38
ecl6ql   Q.Li                  p0   13 Mon 13:01
csc2u0ev CSC203A student 2421  p3      Mon 13:05
csvax1
cbl6nd   N.Drakos              p5   20 Mon 09:57
cblslcd
eclsc    Steve Chidlow         p6   45 Sat 16:32
sgi006
csc6agc  A.G.Cohn              p8 1:04 Mon 10:59
csparc6
ecl6bk   B.Kelly               U2    1 Mon 13:11
```

7 Filestore Commands

In this section a summary of commands which can be used to create and delete files and directories, and move about the filestore are given.

Directory Commands

Creating a Directory

The **mkdir** command is used to create directories. The format of this command is:

```
% mkdir directory_name
```

Jane Brown wants to store her Fortran programs in a directory called `progs` beneath her home directory. In order to create this directory she uses the command:

```
% mkdir progs
```

Deleting a Directory

The **rmdir** command is used to delete (remove) directories. The format of this command is:

```
% rmdir directory_name
```

Jane Brown stores files for project work in a directory called `proj`. When the project has been completed she deletes the directory using the command:

```
% rmdir proj
```

Note that the directory **must** be empty before it can be deleted.

Listing Contents of a Directory

The command **ls** is used to list the contents of a directory. For example:

```
% ls
file1     progs      test.f     test
```

Notice that directories are listed as well as files. To list all files, including hidden files, give the command:

```
% ls -a
.cshrc    file1        progs    test.f     test
```

Hidden files begin with a **.** (a full stop) as illustrated below:

```
% ls -a
.cshrc   .history  .login    .logout
```

The purpose of some of these hidden files is given below.

`.cshrc`	contains commands that are executed every time you start off a C-shell, including when you log in.
`.history`	contains a record of previously executed commands.
`.login`	contains commands that are executed at login time.
`.logout`	contains commands that are executed at logout time.

Warning Hidden files are mainly used to set up and manage your working environment and should not be changed unless you are sure of what you are doing. Some alterations may result in you being unable to login or unable to do much once you have logged in.

To identify directories in a listing give the command:

```
% ls -F
file1       progs/      test.f      test
```

Notice how the directory is identified by the slash (/) character.

File Commands

Deleting Files

Files can be deleted using the **rm** command. For example:

```
% rm file_name        e.g        % rm test.f
```

Displaying Files

The command **cat** is used to display the contents of a file on the screen. For example:

```
% cat file_name        e.g        % cat test.f
```

Creating Files

The command **cat** can also be used to create a file. For example:

```
% cat > file_name
```

For example:

```
% cat > test.f
When typing in a new file
the input must be terminated by
<Ctrl D>
```

Note *Hold the <Ctrl> key down and while holding it down press the D key once then release both the <Ctrl> and D key.*

Note You would normally use an editor for creating files. This example is given since it illustrates how to create a small file without needing to learn the use of an editor.

In the example given above **cat** takes the characters you type and puts them in the file called **test.f**.
The <Ctrl D> key combination tells **cat** that you have finished typing all the required text for **test.f**.

Copying Files

The command **cp** is used to copy a file. The format is:

```
% cp old_file new_file
```

For example:

```
% cp test test.f
```

Renaming Files

The command **mv** is used to rename a file. The format is:

```
% mv old_name new_name
```

Moving Files

The command **mv** is also used to move a file to a new location in the filestore hierarchy. For example:

```
% mv old_name directory
```

For example:

```
% mv test progs/test.f
```

Warning

Some commands (e.g. **rm**, **cp** and **mv**) can be dangerous if not used with care. The command:

```
% cp old_file new_file
```

will delete *new_file* if it exists. If you have spelt the name of *new_file* incorrectly you may accidentally overwrite the contents of another file.
To request a warning if you are in danger of overwriting an existing file you can use the **–i** option with both the **cp** and **mv** command.

For example:

```
% cp –i test test.f
```

Will give the warning
cp: overwrite test.f (yes/no)?
if **test.f** already exists. A similar message will be given with the **mv** command.

Using the wildcard symbol * with the command **rm** can also be dangerous. Wild cards are explained more fully later in this section but the command:

```
% rm test*
```

will delete all files starting with `test`. However if you type an extra space:

```
% rm test *
```

the file **test** will be deleted if it exists. Then all other files in the directory will be deleted! No warning will be given.

To prevent accidental deletion of files you can also use the **-i** option with the **rm** command. The format of the command is:

```
% rm -i file_name
```

You will be asked to confirm that files are to be deleted.
rm: remove file_name (yes/no)?

Note ISS has changed the default for the **rm** command. It will always ask you to confirm that files are to be deleted. However, you are advised not to rely on this as systems not managed by ISS may not have this default set up.
For your own safety it is probably wise to get into the habit of using the **–i** option as a matter of course.

Moving About The Filestore

Changing Working Directory

The directory in which you are located is known as your *working directory*. The command **cd** is used to change your working directory. For example:

```
    % cd directory_name     e.g. % cd progs
```

cd on its own will return you to your home directory.
For example:

```
    % cd
```

Display Working Directory

The command **pwd** is used to display your working directory.
(**pwd** means *print working directory*).
For example:

```
    % pwd
    /home/ecuserv1/men5jb/progs
```

Exercise 2

1. Display your current working directory using the **pwd** command.
2. Create a directory called `exercises`.
3. Change to the directory called `exercises` and display the new current working directory.
4. Return to your home directory.
5. List the contents of your directory. Use the **-l** and **-a** options and compare the output.
6. Move to the directory called `exercises`. Create a file called `example1` using the **cat** command containing your name and address.
7. List the contents of your directory.

Wildcards

Wildcard characters can be used to identify directory and file names. The wildcard character **`*`** is used to refer to any combination of characters. For example:

`% `**`ls *`**	refers to all files
`% `**`cat test*`**	refers to all files starting with `test`

The wildcard character **`?`** is used to refer to a single character. For example:

`% `**`ls test?`**	refers to files starting with `test` followed by a single character
`% `**`cat test.?`**	refers to all files starting with `test` with a single character after the full stop. For example: `test.c` `test.f` `test.1` `test.A`

8 Other Useful UNIX Commands

Many commands are available in the UNIX operating system. This section describes a number of useful commands.

Viewing Files

The following commands can be used to view the contents of a file.

The "more" Command

The command **more** is used to display the contents of a file on the screen. The command is particularly useful for viewing long files since the display stops at the bottom of the screen.

```
% more test.f
c
     program test
     isum = 0
       ...
 10  continue
     print *,"Sum is",isum
c
--More--
(66%)
```

The message at the bottom of the screen means that 66% of the file has been viewed so far. You can now do the following:

- To continue viewing press the space bar
- To view the next line press `<Enter>`
- To quit press the **q** key
- To display the next occurrence of a string of characters type /*string*

For a list of valid commands press the **h** key.

Searching Files

The command **grep** is used to search a file for a string of characters. For example, to search the file *test.f* for the characters *sum*, use the command:

```
% grep sum test.f

     print *,"Sum is",isum
```

A wide range of options can be used with the grep command. In the following example, "^c" shows the use of a *regular expression*. In this example, the search is restricted to lines beginning with the *c* character.

```
% grep "^c"  test.f

c

c    program test
```

Note **grep** stands for *global regular expression* and the **^** symbol in the given *regular expression* indicates *start of line*.

Obtaining Hard Copy Output

The command **`lpr`** sends a file to the default printer:

```
% lpr file1.f
```

Note The command **`lp –d`** is used on some UNIX systems.

Normally a default printer is not defined on ISS systems so a printer identifier must be given.

The locally developed command:

```
% printers
```

can be used on ISS systems to obtain a list of printers.

The command:

```
% lpr -Pprinter file   (or lp –d printer file)
```

is used to submit the file to a specific printer. For example the command:

```
% lpr -Plw1 test.dat    (or lp –d lw1 test.dat )
```

will send the file `test.dat` to the printer `lw1`. Note that no space should be given between the **`-P`** option and the printer name.

Also note that you will normally be charged for output which is sent to a laser printer.

Control Characters

Deleting the Last Character Typed
You can delete the last character typed by pressing the **`<Backspace>`** key. Note that on some UNIX systems the **`<Delete>`** key must be used.

Deleting the Entire Line
If you make many typing mistakes you can delete the entire line by pressing the **`<Ctrl U>`** key.

Sending an Interrupt
If you wish to terminate the execution of a command press **`<Ctrl C>`**.

Sending an End Of File Character
In many UNIX commands you need to finish your input with an end-of-file character. The default end-of-file character is **`<Ctrl D>`**
This was used in the example using *cat* in section 7.

Warning **`<Ctrl S>`** will stop output to the screen, so if you accidentally press **`<Ctrl S>`** your screen will freeze up and it will appear as if the system you are working on has halted.
If you suspect this has happened **`<Ctrl Q>`** can be used to resume output to your screen.

9 File Permissions

The UNIX file security system can prevent unauthorised users from reading or altering files. The file permissions can be displayed using the command:

```
% ls -l [filename]
```

For example, to display the permissions on the file `test.f` type the command:

```
% ls -l test.f
-rw-r--r- 1  men5jb     193  Jul 26  test.f
```

The first set of characters in the output give the permissions:

```
-rw-r--r--
```

The first character specifies the file type. This is normally:

-	to indicate a file
d	to indicate a directory

The remaining groups of characters indicate the permissions for the *owner* of the file, for other users in the same *group* as the owner, and for all *other* users. For example:

```
-rw-r--r--
```

Can be broken down into:

```
file_type   owner group others
-           rw-   r--   r--
```

Indicating that the *owner* has read and write permissions,
users in the same *group* have read permission only
and all *other* users have read permission only.

Changing File Permissions

The command **chmod** is used to change the permissions on a file. The format of this command is:

```
% chmod mode filename
```

For example, if the file `test.f` has read and write permissions for the owner and no access permissions for group or others (**-rw-------**) and you wish other members of the same group to read the file you would give the command:

```
% chmod g+r test.f
```

Resulting in permissions of **-rw-r-----**

If execute permission was later required by the owner the command would be:

```
% chmod u+x test.f
```

Resulting in permissions of **-rwxr-----**

chmod Modes

In the command:

```
% chmod mode filename
```

the mode consists of three components:

- who
- operator
- permissions

The following options are possible:

who options

u	user (owner)
g	group
o	other
a	all

permissions

r	read
w	write
x	execute

operators

-	remove permission
+	add permission
=	assign permission

For example:

`o-rw` removes `r` and `w` rights from others:
`chmod o-rw file1.f`

`u+x` adds `x` (execute) permission to the owner:
`chmod u+x test`

Exercise 3

1. Create a file called `list` containing the following lines:

   ```
   pwd
   ls -l
   ```

2. Display the permissions on the file called `list` using the command **`ls -l`**
3. Use the **`chmod`** command to give you (the user) execute permissions (`x`) on the file.
4. Execute the file by typing **`list`**

10 Getting Help

The command **man** is used to display help on the syntax and use of UNIX commands. (**man** is short for manual i.e. you are accessing the *online manual pages*)

The format of this command is:

```
% man [option] [command|keyword]
```

For example to obtain help information on the **who** command type:

```
% man who
```

The keyword option **-k** *keyword* is used to display a list of help files associated with the keyword. For example to display a list of all `man` files associated with `password` type the command:

```
% man -k password

getpass(3)     read a password

passwd(1)      change login password

passwd(4)      password file
```

If information is contained in more than one section of the online manual, as in the above example for *passwd*, you can access a specific section by using the **–s** option.

```
e.g.  % man –s 1 passwd
or    % man –s 4 passwd
```

Reading man Pages

The command **man** automatically invokes the **more** program for viewing files. When a complete screen of information has been displayed the message:

```
--
More
--
```

will be displayed. You can use the normal **more** commands to continue viewing.

Exercise 4

1. Use the **man** command to find out about the **who** command.

2. Use the **man** command to find out about the **ls** command and identify the **-a** and **-l** options used previously.

11 Standard Input and Output

Standard Input

Input to UNIX commands is often given from the keyboard. For example consider the **spell** command, which is the UNIX spelling checker:

```
% spell
Input to the spell ulitity
is typed at the keyboard
<Ctrl/D>
```

Once input to **spell** is terminated, it will echo back any words it detects as being spelled incorrectly.

Standard Output

Output from UNIX commands is normally displayed on the screen. For example:

```
% spell
Input to the spell ulitity
is typed at the keyboard
<Ctrl D>
ulitity                        Displayed on screen
```

Re-direction of Standard Input

It is possible to redirect standard input so that the input is taken from a file. Imagine you wish to check for spelling errors in a report. The contents of the file `report` can be fed into the **spell** command:

Input to UNIX commands is normally given from the keyboard. For example:

```
% spell < report
ulitity
```

The **<** character is used to re-direct the input from the file `report` to the command **spell**. The general format for re-direction of user input is:

```
command < filename
```

Another common use of re-direction of standard input is to mail a file to another user. The command:

```
% elm J.Smith@leeds.ac.uk < report
```

will mail the contents of the file **report** to `J.Smith`

Re-direction of Standard Output

You do not always want the output from a UNIX command to be displayed on the screen. Imagine you want a list of your files and directories kept in a file. You can use the command:

```
% ls > filelist
```

The **>** character is used to re-direct the output from the command to the file called `filelist`. The general format for re-direction of user output is:

```
command > filename
```

Note that if the file `/dev/null` is given, output from the command is discarded.

Re-direction of Input and Output

It is possible to re-direct both standard input and output. If you have a report containing many spelling mistakes you may wish to keep a list of the mistakes in a file. You can do this using the following command:

```
% spell < report > errors
```

Exercise 5

1. Create a file called `filelist` containing a list of files in your directory by redirecting the output from the **`ls`** command.

2. Use the **cat** command to viev the contents of the file `filelist`.

12 Piping

Output from one command can be piped to the input of another command using the pipe (**|**) character:

```
command1 | command2
```

If you wish to send a listing of your files and directories to a printer you could pipe the output of the **`ls`** command to the **`lpr`** command which sends the list to the printer ps4:

```
% ls | lpr –Pps4
```

To give an illustration of the use of a pipe, imagine you wish to list all sub-directories in your current working directory. The command:

```
% ls -l
```

lists all files and directories, identifying directories by the character `d` at the start of each line. You can obtain a list of directories only by piping the output of this command to the **`grep`** command, giving **`grep`** an option which will list only lines containing the **`d`** character at the start of the line. The command is:

```
% ls -l | grep "^d"
```

The commands **`sort`** and **`grep`** are often used when piping. For example:

```
% cat phonenos | sort
```

will sort a list of phone numbers held in the file **phonenos**

```
% cat phonenos | sort | wc -l
```

will sort a list of phone numbers in the file **phonenos** and tell you how many lines are contained in that sorted list.

```
% cat phonenos | grep leeds | sort | lpr –Pps4
```

will send a sorted list of phone numbers containing the characters **`leeds`** to the printer ps4.

Note **wc** is the *word count* command. With the **-l** option it will return the number of lines.

Exercise 6

1. Type the command **`ls -l`** and examine the format of the output.

2. Pipe the output of the command **`ls -l`** to the word count program **`wc`** to obtain a count of the number of files in your directory.